

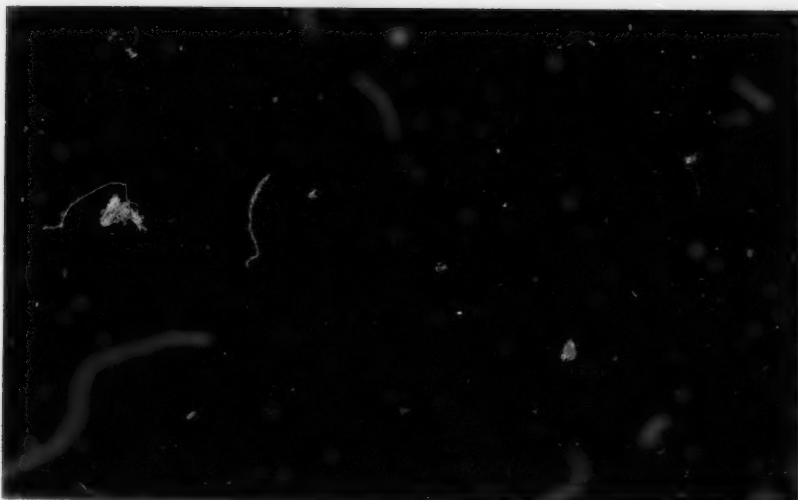


**CENTRE DE RECHERCHE
SUR LES TRANSPORTS**

(C.R.T.)

**CENTRE FOR RESEARCH
ON TRANSPORTATION**

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE



CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A,
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE: (514) 343-7575
TÉLEX: (21) 05-561325
EMAIL: crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX): (514) 343-7121

CRT-715

**A Shortest Path Algorithm for Networks
with Time Dependent Link
Travel Times**

**by
Omar Drissi-Kaïtouni**

This research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

Centre de recherche sur les transports — Publication #715

December 1990

ABSTRACT

This paper describes an efficient shortest path algorithm for networks that have time dependent link travel times (e.g., dynamic networks).

The algorithm that we propose is an extension of the popular Dijkstra's shortest path algorithm, that takes into account the arrival time of flows at nodes in order to handle the time dependent link travel times. The proposed algorithm keeps the logarithmic complexity of Dijkstra's algorithm (when using heaps) plus a linear part related to the study horizon (number of periods of time).

Key words: Shortest Path, Dynamic, Traffic, Network, Assignment.

RÉSUMÉ

Cet article décrit un algorithme efficace pour le calcul des chemins les plus courts sur un réseau où les coûts et/ou les temps de parcours sur les arcs dépendent du temps (i.e., réseaux dynamiques).

L'algorithme que nous décrivons est une extension de l'algorithme de Dijkstra, où nous tenons compte des temps d'arrivée aux noeuds pour définir les coûts et/ou les temps de parcours sur les arcs. La complexité de l'algorithme proposé comprend une partie qui croît de façon logarithmique avec la taille du réseau, et une partie qui croît de façon linéaire avec le nombre de périodes de temps d'étude.

Mots clés: Chemin le plus court, dynamique, trafic, réseau, affectation.

1. INTRODUCTION

This paper describes an efficient shortest path algorithm for networks that have time dependent link travel times (e.g. dynamic networks).

Given a network $G = (N, A)$, where N is the set of nodes, and A the set of arcs, with each arc $a \in A$ is associated a travel time τ_a^t for every period of time $t = 1, \dots, T$. We assume that the arcs have an unlimited storage capacity. The problem is to find the tree of minimum travel time paths from a root node $r \in N$ to a set of destination nodes $D \subseteq N$.

An impressive number of papers has been published on algorithms that solve the "static" shortest path problem, e.g., when each link is associated with one travel time and/or cost value. The most popular algorithm to solve this problem, when the link costs are nonnegative, is Dijkstra's labeling algorithm [6] that has very efficient implementations (see for instance Dial et al. [5], Denardo and Fox [4], Ahuja et al. [1], Mondou et al. [15]).

The algorithm that we propose is an extension of the popular Dijkstra's shortest path algorithm, that takes into account the arrival time of flows at nodes in order to handle the time dependent link travel times.

We think that this contribution may be very useful for solving many dynamic network flow problems. For instance, this algorithm may be an efficient tool to solve the subproblems of the linear approximation algorithm related to the nonlinear dynamic traffic assignment problem (Zawack and Thompson [18], Drissi and Hameda [7]) where at each iteration, the demand must be assigned on the shortest paths of a network that has link time dependent travel times. The algorithm is sufficiently general to be applied to other dynamic network flow problems.

The remainder of this paper is organized as follows: in section 2 we describe the algorithm and evaluate its complexity. In section 3, we give a numerical example, and we conclude the paper in section 4.

2. SHORTEST PATH ALGORITHM FOR DYNAMIC NETWORKS

The notation that will be used is the following:

- A : set of arcs of the network.
- N : set of nodes of the network.
- A_l : set of arcs having node l as tail node (e.g., $(l, j) \in A$).
- T : number of periods of time.
- τ_a^t : travel time on arc a during period of time t .
- r : root node.
- D : set of destination nodes.

We assume that the travel times τ_a^t , $a \in A$, $t = 1, \dots, T$, are nonnegative and are expressed in the same units as the periods of time.

The problem that we seek to solve in the following, is to find the tree of minimum travel time paths from the root node r to a set of destinations $D \subseteq N$. The most trivial solution process that may be used is to apply the classical Dijkstra's algorithm

on a time-space expansion of the network (see Ford and Fulkerson [9], Fulkerson [10], Zawack and Thompson [18], Drissi and Hameda [7]). Figure 1 illustrates an instance of time-space expansion of a simple network. The size of the expanded network is about $T \cdot |N|$ nodes and $2 \cdot T \cdot |A|$ arcs. Therefore, the use of the classical Dijkstra's shortest path algorithm on the expanded network would lead to a complexity of about $O((T \cdot |N| + 2 \cdot T \cdot |A|) \log(T \cdot |N|))$ or $O((T \cdot (|N| + |A|)) \log(T \cdot |N|))$ (if Dijkstra's algorithm is implemented with heaps).

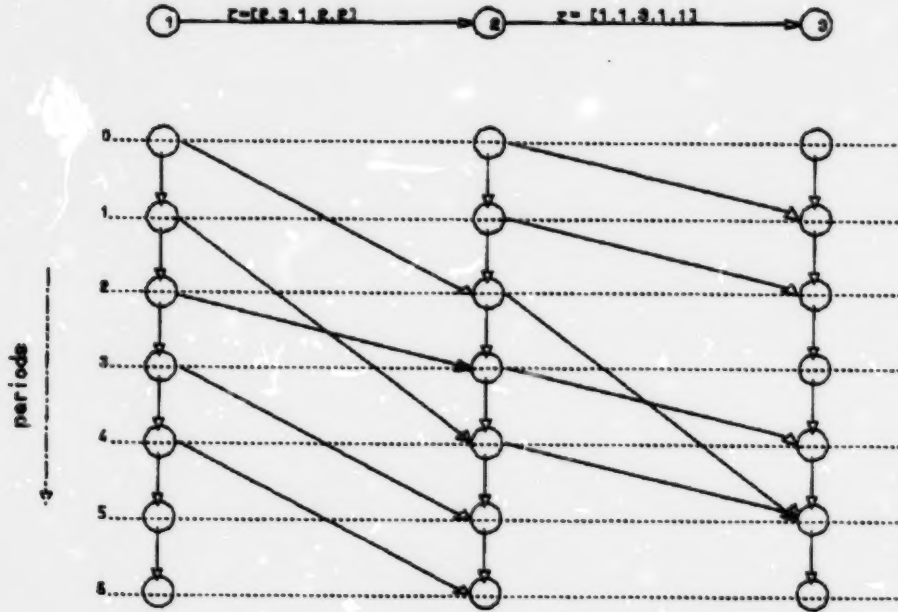


Figure 1: Expansion of a simple network

In the following, we show that it is possible to use Dijkstra's algorithm on the original network (hence more efficiently) by using modified link travel times instead of the original link travel times. Let's define the modified, time dependent, link travel time $\Delta_a^{\delta_i}$ to be:

$$\Delta_a^{\delta_i} = \min\{\tau_a^t + t - \delta_i : t \geq \delta_i\} \quad \forall a = (i, j) \in A, \quad \forall \delta_i = 1, \dots, T.$$

In this expression, $\tau_a^t + t - \delta_i$, for $t \geq \delta_i$, represents the total time required to cross link a if the arrival time at the tail node i of link a is δ_i , and that the effective entry on link a is delayed by $t - \delta_i$ periods; in the expanded network, this delay corresponds to the use of $t - \delta_i$ vertical links. Therefore, $\Delta_a^{\delta_i}$ represents the best total time to cross link a , when the arrival time at its tail node i is δ_i . Now, in the labeling process of Dijkstra's algorithm, the arrival time δ_i to a node i that has received a permanent

label is also the minimum travel time from the root node r to node i (hence will not be modified later). Therefore, as soon as a node i has received a permanent label, $\Delta_a^{\delta_i}$ is the best total time to cross link $a = (i, j)$ under the constraint that the flows initially at the root r at period δ_r have already arrived to node i at period δ_i . Hence, the algorithm takes the following form:

Dynamic Shortest Path Algorithm (A1)

Step 0- Initialization

- $\delta_l = \infty \forall l \in N$ (Minimum travel time from r to all other nodes is initialized to ∞)
- $\delta_r = 0$
- $\Omega = \{r\}$ (Set of temporary labeled nodes, e.g., reached but not yet scanned)
- $p_l = \text{nil} \forall l \in N$ (Predecessor node of l in the tree of shortest paths is initially unknown)

Step 1- Stopping Criterion

If $\Omega = \emptyset$ then Stop

Step 2- Find node i to be scanned (node to be permanently labeled)

- i- Let i be the node for which $\delta_i = \min\{\delta_l : l \in \Omega\}$
- ii- Let $\Omega = \Omega - \{i\}$

Step 3- Scan node i For every link $a = (i, j) \in A_i$ do:

- i- Let $\Delta_a^{\delta_i} = \min\{\tau_a^t + t - \delta_i : t \geq \delta_i\}$
- ii- If $\delta_i + \Delta_a^{\delta_i} < \delta_j$ then
 - α - Let $\delta_j = \delta_i + \Delta_a^{\delta_i}$
 - β - Let $p_j = i$
 - γ - Let $\Omega = \Omega \cup \{j\}$

We must note that the stopping criterion used in Step 1- leads to find the tree of shortest paths from the root to all the other nodes. If only the tree of shortest paths from the root to a set of destination nodes D is required, the following stopping criterion may be used: "If all $q \in D$ have already been scanned, then Stop".

Also, in the initialization Step 0-, by setting $\delta_r = 0$, we suppose that the starting period of time at origin node r is zero; however, if the starting period of time is not 0, it suffices to set δ_r to the required value.

Now let's examine the complexity of this algorithm: the complexity of Step 0-, which is executed only once, is $O(|N|)$. Complexity of Step 1-, which is executed $|N|$ times is $O(|N|)$ too. As for the complexity of Step 2-, it depends on the structure of

Ω ; if Ω is organized as a heap, searching or deleting an element from the heap can be done in $O(\log|N|)$; hence, since this step is executed $|N|$ times, its complexity is $O(|N|. \log|N|)$. In Step 3-, the computation of $\Delta_a^{\delta_i}$ can be done in $O(T)$, and since this is done only once for each arc, then complexity of Step 3-i- is $O(T.|A|)$; adding an element to the heap Ω can be done in $O(\log|N|)$, and since Step 3-ii- is executed $|A|$ times, its complexity is $O(|A|. \log|N|)$. Therefore the complexity of the whole algorithm is $O((|N| + |A|). \log|N| + T.|A|)$. This complexity is obviously much better than the one obtained by applying Dijkstra's algorithm to a time-space expanded network, that is $O((T.(|N| + |A|). \log(T.|N|)))$.

3. NUMERICAL EXAMPLE

Consider the network depicted in figure 2, where the travel times are as follows:

periods t	[	0	1	2	3	4	5	6	7	8	9	10	11	12	...
links a															
1	[	2	1	1	1	1	3	3	3	3	3	3	3	3	...
2	[	5	3	1	1	1	1	1	1	1	1	1	1	1	...
3	[	1	2	3	1	2	3	1	2	3	1	2	3	1	...
4	[	6	4	6	4	6	4	6	4	6	4	6	4	6	...
5	[	2	2	2	2	2	2	1	1	1	1	1	1	1	...
6	[	1	1	1	1	1	1	2	2	2	2	2	2	2	...
7	[	2	2	2	2	2	2	1	1	1	1	1	1	1	...
8	[	1	1	1	1	1	1	1	1	1	1	1	1	1	...
9	[	6	6	6	6	5	3	1	1	1	2	2	2	2	...

Suppose that we want to find the tree of shortest paths from the origin node 1 to all the other nodes starting at the time period 0. The shortest paths obtained by algorithm A1 are reported in dark line in figure 2 where we also give for each node the minimal arrival times and its predecessor in the tree of shortest paths. Then we give the computation details of algorithm A1 to solve this problem.

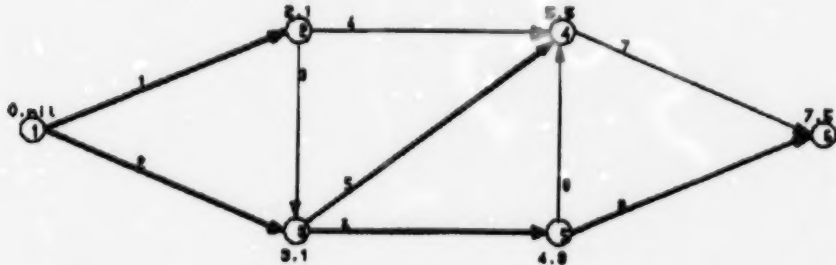


Figure 2: Example network

The algorithm starts with $\delta_1 = 0$, and $\delta_l = \infty, l = 2, \dots, 6$, $\Omega = \{1\}$ and $p_l = \text{nil}, l = 1, \dots, 6$.

First iteration:

Step 1: $\Omega \neq \emptyset$; continue.

Step 2: $\min\{\delta_l : l \in \Omega\} = \delta_1$; so $i = 1$ and $\Omega = \Omega - \{1\} = \emptyset$.

Step 3: Scan node 1:

* $a = 1 = (1, 2)$: $\Delta_a^{\delta_1} = \min\{\tau_a^t + t - \delta_1 : t \geq \delta_1 = 0\} = \min\{2, 2, 3, 4, 5, \dots\} = 2$, for $t = 0$. Therefore:

$\delta_1 + \Delta_a^{\delta_1} = 0 + 2 = 2 < \delta_2 = \infty$, then:

$\delta_2 = 2, p_2 = 1$ and $\Omega = \Omega \cup \{2\} = \{2\}$

* $a = 2 = (1, 3)$: $\Delta_a^{\delta_1} = \min\{\tau_a^t + t - \delta_1 : t \geq \delta_1 = 0\} = \min\{5, 4, 3, 4, 5, 6, 7, \dots\} = 3$, for $t = 2$. Therefore:

$\delta_1 + \Delta_a^{\delta_1} = 0 + 3 = 3 < \delta_3 = \infty$, then:

$\delta_3 = 3, p_3 = 1$ and $\Omega = \Omega \cup \{3\} = \{2, 3\}$

Second iteration:

Step 1: $\Omega \neq \emptyset$; continue.

Step 2: $\min\{\delta_l : l \in \Omega\} = \delta_2$; so $i = 2$ and $\Omega = \Omega - \{2\} = \{3\}$.

Step 3: Scan node 2:

* $a = 3 = (2, 3)$: $\Delta_a^{\delta_2} = \min\{\tau_a^t + t - \delta_2 : t \geq \delta_2 = 2\} = \min\{-, -, 3, 2, 4, 6, 5, \dots\} = 2$, for $t = 3$. Therefore:

$\delta_2 + \Delta_a^{\delta_2} = 2 + 2 = 4 \geq \delta_3 = 3$, then nothing to do.

* $a = 4 = (2, 4)$: $\Delta_a^{\delta_2} = \min\{\tau_a^t + t - \delta_2 : t \geq \delta_2 = 2\} = \min\{-, -, 6, 5, 8, 7, 10, \dots\} = 5$, for $t = 3$. Therefore:

$\delta_2 + \Delta_a^{\delta_2} = 2 + 5 = 8 < \delta_4 = \infty$, then:

$\delta_4 = 8, p_4 = 2$ and $\Omega = \Omega \cup \{4\} = \{3, 4\}$

Third iteration:

Step 1: $\Omega \neq \emptyset$; continue.

Step 2: $\min\{\delta_l : l \in \Omega\} = \delta_3$; so $i = 3$ and $\Omega = \Omega - \{3\} = \{4\}$.

Step 3: Scan node 3:

* $a = 5 = (3, 4)$: $\Delta_a^{\delta_3} = \min\{\tau_a^t + t - \delta_3 : t \geq \delta_3 = 3\} = \min\{-, -, -, 2, 3, 4, 4, 5, 6, \dots\} = 2$, for $t = 3$. Therefore:

$\delta_3 + \Delta_a^{\delta_3} = 3 + 2 = 5 < \delta_4 = 8$, then:

$\delta_4 = 5, p_4 = 3$ and $\Omega = \Omega \cup \{4\} = \{4\}$

* $a = 6 = (3, 5)$: $\Delta_a^{\delta_3} = \min\{\tau_a^t + t - \delta_3 : t \geq \delta_3 = 3\} = \min\{-, -, -, 1, 2, 3, 5, \dots\} = 1$, for $t = 3$. Therefore:

$\delta_3 + \Delta_a^{\delta_3} = 3 + 1 = 4 < \delta_5 = \infty$, then:

$\delta_5 = 4, p_5 = 3$ and $\Omega = \Omega \cup \{5\} = \{4, 5\}$

Fourth iteration:

Step 1: $\Omega \neq \emptyset$; continue.

Step 2: $\min\{\delta_l : l \in \Omega\} = \delta_5$; so $i = 5$ and $\Omega = \Omega - \{5\} = \{4\}$.

Step 3: Scan node 5:

* $a = 8 = (5, 4)$: $\Delta_a^{\delta_5} = \min\{\tau_a^t + t - \delta_5 : t \geq \delta_5 = 4\} = \min\{-, -, -, -, 1, 2, 3, 4, 5, \dots\} = 1$, for $t = 4$. Therefore:

$\delta_5 + \Delta_a^{\delta_5} = 4 + 1 = 5 \geq \delta_4 = 5$, then nothing to do.

* $a = 9 = (5, 6)$: $\Delta_a^{\delta_5} = \min\{\tau_a^t + t - \delta_5 : t \geq \delta_5 = 4\} = \min\{-, -, -, -, 5, 4, 3, 4, 5, \dots\} = 3$, for $t = 7$. Therefore:

$\delta_5 + \Delta_a^{\delta_5} = 4 + 3 = 7 < \delta_6 = \infty$, then:

$\delta_6 = 7$, $p_6 = 7$ and $\Omega = \Omega \cup \{6\} = \{4, 6\}$

Fifth iteration:

Step 1: $\Omega \neq \emptyset$; continue.

Step 2: $\min\{\delta_l : l \in \Omega\} = \delta_4$; so $i = 4$ and $\Omega = \Omega - \{4\} = \{6\}$.

Step 3: Scan node 4:

* $a = 7 = (4, 6)$: $\Delta_a^{\delta_4} = \min\{\tau_a^t + t - \delta_4 : t \geq \delta_4 = 5\} = \min\{-, -, -, -, -, 2, 3, 4, 5, \dots\} = 3$, for $t = 5$. Therefore:

$\delta_4 + \Delta_a^{\delta_4} = 5 + 3 = 8 \geq \delta_6 = 7$, then nothing to do.

Sixth iteration:

Step 1: $\Omega \neq \emptyset$; continue.

Step 2: $\min\{\delta_l : l \in \Omega\} = \delta_6$; so $i = 6$ and $\Omega = \Omega - \{6\} = \emptyset$.

Step 3: Scan node 6: no link exits from node 6, then nothing to do.

Seventh iteration:

Step 1: $\Omega = \emptyset$; stop.

4. CONCLUSION

This paper describes an efficient shortest path algorithm for networks that have time dependent link costs and/or travel times (e.g. dynamic networks); the problem is to find the tree of minimum travel time paths from a root node in the network to a set of destination nodes in the network. The algorithm that we propose is an extension of Dijkstra's popular shortest path algorithm, which takes into account the arrival time of flows at nodes in order to handle the time dependent link costs and/or travel times. The complexity of the algorithm grows logarithmically with the size of the network and linearly with the number of periods of time.

We believe that this contribution may be very useful for solving many dynamic network flow problems.

REFERENCES

- [1] Ahuja, R.K., Mehlhorn, K., Orlin, J.B. and Tarjan, R.E. (1988) Faster Algorithms for the Shortest Path Problem, Technical Report No. 193, Operations Research Center, M.I.T..
- [2] Arnott, R., de Palma, A. and Lindsey, R. (1987) Departure Time and Route Choice for Routes in Parallel, Discussion Paper #658. Institute for Economic Research, Queen's University.
- [3] Carey, M. (1987) Optimal Time-Varying Flows on Congested Networks, *Operations Research* 35 No.1, 58-69.
- [4] Denardo, E.V. and Fox, B.L. (1979) Shortest-Route Methods: 1. Reaching, Pruning and Buckets, *Operations Research* 27, 161-186.

- [5] Dial, B., Glover, F., Karney, D. and Klingman, D. (1979) A Computational Analysis of Alternatives Algorithms and Labeling Techniques for Finding Shortest Path Trees, *Networks* 9, 215-248.
- [6] Dijkstra, E.W. (1959) A Note on Two Problems in Connexion with Graphs, *Numerische Mathematik* 1.
- [7] Drissi-Kaïtouni, O. and Hameda-Benchekroun, A. (1989) A Dynamic Traffic Assignment Model and a Solution Algorithm, publication #649, Centre de recherche sur les transports, Université de Montréal.
- [8] Drissi-Kaïtouni, O. (1990) A Model for the Dynamic Traffic Problem, publication #702, Centre de recherche sur les transports, Université de Montréal.
- [9] Ford, L.R. and Fulkerson, D.R. (1962) *Flows in Networks*, Princeton University Press. Princeton, New Jersey.
- [10] Fulkerson, D.R. (1975) Studies in Graph Theory, part I, *Studies in Mathematics* 11, The Mathematical Association of America.
- [11] Hendrickson, C. and Kocur, G. (1981) Schedule Delay and Departure Time Decision in a Deterministic Model, *Transportation Science* 15, 62-77
- [12] Ho, J.K. (1980) A Successive Linear Optimization Approach to the Dynamic Traffic Assignment Problem, *Transportation Science* 14 No.4, 295-305.
- [13] Merchant, D.K. and Nemhauser, G.L. (1978) A Model and an Algorithm for the Dynamic Traffic Assignment Problem, *Transportation Science* 12 No.3, 183-199.
- [14] Merchant, D.K. and Nemhauser, G.L. (1978) Optimality Conditions for a Dynamic Traffic Assignment Model, *Transportation Science* 12 No.3, 200-207.
- [15] Mondou, J.F., Crainic, T.G. and Nguyen, S. (1990) Shortest Path Algorithms: A Computational Study with the C Programming Language, Publication No. 676, Centre de recherche sur les transports, Université de Montréal.
- [16] Wardrop, J.G. (1952) Some Theoretical Aspects of Road Traffic Research, *Proceedings of the Institut of Civil Engineers*, Part II 1, 325-378.
- [17] Yagar, S. (1970) Dynamic Traffic Assignment by Individual Path Minimization and Queuing, *Transportation Research* 5, 179-196.
- [18] Zawack, D.J. and Thompson, G.L. (1987) A Dynamic Space-Time Network Flow Model for City Traffic Congestion, *Transportation Science* 21 No.3, 153-162.